intel®

Intel® Itanium® 2 Processor Reference Manual

For Software Development and Optimization

May 2004

Order Number: 251110-003

There are six general-purpose ALU units (ALU0, 1, 2, 3, 4, 5), two integer units (I0, 1), and one shift unit (ISHIFT, used for general purpose shifts and other special instructions). A maximum of six of these types of instructions can be issued per cycle.

The Data Cache Unit (DCU) contains four memory ports. Two ports are generally used for load operations; two are generally used for store operations. A maximum of four of these types of instructions can be issued per cycle. The two store ports can support a special subset of the floating-point load instructions.

There are six multimedia functional units (PALU0, 1, 2, 3, 4, 5), two parallel shift units (PSMU0, 1), one parallel multiply unit (PMUL), and one population count unit (POPCNT). These handle multimedia, parallel multiply, and the `popcnt` instruction types. At most, one `pmul` or `popcnt` instruction may be issued per cycle. However, the Itanium 2 processor may issue up to six PALU instructions per cycle.

There are four floating-point functional units: two FMAC units to execute floating-point multiply-adds and two FMISC units to perform other floating-point operations, such as `fcmp`, `fmerge`, etc. A maximum of two floating-point operations can be executed per cycle.

There are three branch units enabling three branches to be executed per cycle.

All of the computational functional units are fully pipelined, so each functional unit can accept one new instruction per clock cycle in the absence of other types of stalls. System instructions and access to system registers may be an exception.

3.3 Instruction Slot to Functional Unit Mapping

Each fetched instruction is assigned to a functional unit through an issue port. The numerous functional units share a smaller number of issue ports. There are 11 issue ports: eight for non-branch instructions and three for branch instructions. They are labeled M0, M1, M2, M3, I0, I1, F0, F1, B0, B1, and B2. The process of mapping instructions within bundles to functional units is called *dispersal*.

An instruction's type and position within the issue group define to which issue port the instruction is assigned. An instruction is mapped to a subset of the issue ports based upon the instruction type (i.e., ALU, Memory, Integer, etc.). Then, based on the position of the instruction within the instruction group presented for dispersal, the instruction is mapped to a particular issue port within that subset.

Table 3-1, "A-Type Instruction Port Mapping," Table 3-2, "I-Type Instruction Port Mapping," and Table 3-3, "M-Type Instruction Port Mapping" show the mappings of instruction types to ports and functional units. Section 3.3.2 describes the selection of the particular port based upon instruction position.

Note: Shading in the following tables indicates the instruction type can be issued on the port(s).

A-type instructions can be issued on all M and I ports (M0-M3 and I0 and I1). I-type instructions can only issue to I0 or I1. The I ports are asymmetric so some I-type instructions can only issue on port I0. M ports have many asymmetries: some M-type instructions can issue on all ports; some can only issue on M0 and M1; some can only issue on M2 and M3; some can only issue on M0; some can only issue on M2.

Table 3-1. A-Type Instruction Port Mapping

Instruction Type	Description	Examples	Ports
A1-A5	ALU	add, shladd	M0-M3, I0, I1
A4, A5	Add Immediate	addp4, addl	M0-M3, I0, I1
A6, A7, A8	Compare	cmp, cmp4	M0-M3, I0, I1
A9	MM ALU	pcmp[1 | 2 | 4]	M0-M3, I0, I1
A10	MM Shift and Add	pshladd2	M0-M3, I0, I1

Table 3-2. I-Type Instruction Port Mapping

Instruction Type	Description	Examples	I Port	
			I0	I1
I1	MM Multiply/Shift	pmpy2.[l | r], pmpyshr2{.u}	▨	
I2	MM Mix/Pack	mix[1 | 2 | 4].[l | r pmin, pmax	▨	▨
I3, I4	MM Mux	mux1, mux2	▨	▨
I5	Variable Right Shift	shr{.u] =ar,ar pshr[2 | 4] =ar,ar	▨	▨
I6	MM Right Shift Fixed	pshr[2 | 4] =ar,c	▨	▨
I7	Variable Left Shift	shl{.u] =ar,ar pshl[2 | 4] =ar,ar	▨	▨
I8	MM Left Shift Fixed	pshl[2 | 4] =ar,c	▨	▨
I9	MM Popcount	popcnt	▨	
I10	Shift Right Pair	shrp	▨	
I11-I17	Extr, Dep Test Nat	extr{.u}, dep{.z} tnat	▨	
I19	Break, Nop	break.i, nop.i	▨	▨
I20	Integer Speculation Check	chk.s.i	▨	▨
I21-28	Move to/from BR/PR/IP/AR	mov =[br | pr | ip | ar] mov [br | pr | ip | ar]=	▨	
I29	Sxt/Zxt/Czx	sxt, zxt, czx	▨	▨

Table 3-3. M-Type Instruction Port Mapping

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
M1, 2, 3	Integer Load	ld*sz*, ld8.fill	▨	▨		
M4, 5	Integer Store	st*sz*, st8.spill			▨	▨
M6, 7, 8	Floating-point Load	ldf*fsz*, ldf*fsz*.s, ldf.fill	▨	▨	▨	▨
	Floating-point Advanced Load	ldf*fsz*.a, ldf*fsz*.c.[clr | nc]	▨	▨		
M9, 10	Floating-point Store	stf*fsz*, stf.spill			▨	▨

Table 3-3. M-Type Instruction Port Mapping (Continued)

Instruction Type	Description	Examples	Memory Port			
			M0	M1	M2	M3
M11, 12	Floating-point Load Pair	ldfp*fsz*	▨	▨		
M13, 14, 15	Line Prefetch	lfetch	▨	▨		
M16	Compare and Exchange	cmpxchg*sz*.[acq | rel]			▨	▨
M17	Fetch and Add	fetchadd*sz*.[acq | rel]			▨	▨
M18	Set Floating-point Reg	setf.[s | d | exp | sig}			▨	▨
M19	Get Floating-point Reg	getf.[s | d | exp | sig}			▨	
M20, 21	Speculation Check	chk.s{.m}			▨	▨
M22, 23	Advanced Load Check	chk.a[clr | nc]	▨	▨		
M24	Invalidate ALAT	invala	▨	▨		
	Mem Fence, Sync, Serialize	fwb, mf{.a}, srlz.[d | i], sync.li	▨			
M25	RSE Control	flushrs, loadrs	▨			
M26, 27	Invalidate ALAT	invala.e	▨	▨		
M28	Flush Cache, Purge TC Entry	fc, ptc.e			▨	
M29, 30, 31	Move to/from App Reg	mov{.m} *ar*= mov{.m} =*ar*			▨	
M32, 33	Move to/from Control Reg	mov *cr*=, mov =*cr*			▨	
M34	Allocate Register Stack Frame	alloc			▨	
M35, 36	Move to/from Proc. Status Reg	mov psr.[l | um] mov =psr.[l | m]			▨	
M37	Break, Nop.m	break.m, nop.m	▨	▨	▨	▨
M38, 39, 40	Probe Access	probe.[r | w].{fault}			▨	
M41	Insert Translation Cache	itc.[d | i]			▨	
M42, 43	Move Indirect Reg Insert TR	mov *ireg*=, move =*ireg*, itr.[d | i]			▨	
M44	Set/Reset User/System Mask	sum, rum, ssm, rsm			▨	
M45	Purge Translation Cache/Reg	ptc.[d | i | g | ga]			▨	
M46	Virtual Address Translation	tak, thash, tpa, ttag			▨	

3.3.1 Execution Width

When dispersing instructions to functional units, the Itanium 2 processor views, at most, two bundles at a time with no special alignment requirements. This text refers to these bundles as the *first* and *second* bundles. A *bundle rotation* causes new bundles to be brought into the two-bundle window of instructions being considered for issue. Bundle rotations occur when all the instructions within a bundle are issued. Either one or two bundles can be rotated depending on how many instructions were issued.

intel®

Intel® Itanium(TM) Architecture Assembly Language Reference Guide

2000 - 2001

Order Number: 248801-004

World Wide Web: http://developer.intel.com

Include File Directive

To include the content of another file in the current file, use the .include directive (see Preprocessor Support) or use the #include directive of the standard C preprocessor.

To include the contents of another file in the current source file, use the .include directive in the following format:

```
.include "filename"
```

Where:

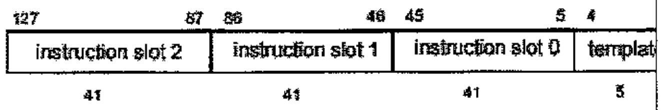
"filename"	Specifies a string constant. If the specified filename is an absolute pathname, the file is included. If the specified filename is a relative pathname, the assembler performs a platform-dependent search to locate the include file.

Bundles

Itanium(TM) architecture instructions are grouped together in 128-bit aligned containers called bundles. Each bundle contains three 41-bit instruction slots, and a 5-bit template field. The template field specifies which type of execution unit processes each instruction in the bundle. Bit 0 is set to 1 if there is a stop at the end of a bundle. There is no fixed relation between the boundaries of an instruction group and the boundaries of a bundle.

Figure below illustrates the format of a bundle.

Bundle Format



Multiway branch bundles contain more than one branch instruction. When the first branch instruction of a multiway bundle is taken, the subsequent branch instruction does not execute.

Bundles are always aligned at 16-byte boundaries. The assembler automatically aligns sections containing bundles to at least 16-bytes.

Bundling can be:

- implicit (automatically performed by the assembler)
- explicit (specified by the programmer)

— with automatic selection of the template

— with explicit selection of the template

Refer to the *Intel® Itanium™ Architecture Software Developer's Manual* for more details about bundles.

Implicit Bundling

The assembler bundles instructions automatically by default.

In the implicit-bundling mode, section directives do not terminate a partially-filled bundle of a previously-defined section. This means that the assembler can return to the previous section and continue to fill the bundle.

In implicit-bundling mode, a label forces the assembler to start a new bundle.

Explicit Bundling

The programmer can explicitly assemble bundles by grouping together up to three instructions, and enclosing them in braces ({}). The assembler places these instructions in one bundle, separate from all preceding and subsequent instructions. Stops at the end of an explicit bundle can be placed before or after the closing brace.

Section directives and data allocation statements cannot be used within an explicit bundle. Cross-section data allocation statements can be used within an explicit bundle. See the Cross-section Data Allocation Statements section for more information.

In explicit-bundling mode, labels can be inserted only as the first statement of an explicit bundle. Instruction tags can be applied to any instruction.

When using explicit-bundling, the appropriate template can be selected in one of the following ways:

- automatically by the assembler.
- explicitly by the programmer, using the explicit-template directives.

Auto-template Selection

By default, the assembler searches and selects a matching template for a bundle. The template fields specify intra-bundle instruction stops. When two templates consist of the same sequence of instruction types, they are distinguished by stops. The assembler selects the appropriate template field based on the stops within the bundle. If no template is found, the assembler produces a diagnostic message. Instruction group stops may occur in a bundle.

Explicit Template Selection

To explicitly select a specific template, use one of the directives listed in table Explicit Template Selection Directive (below) as the first statement of your code within the braces. For example, the `.mii` directive selects the memory-integer-integer (`mii`) template.

Explicit Template Selection Directives

Directive		**Template Selection**	
	Slot 0	**Slot 1**	**Slot 2**
`.mmi`	memory	integer	integer
`.mfi`	memory	floating point	integer
`.bbb`	branch	branch	branch
`.mlx`	memory	long immediate	
`.mib`	memory	integer	branch
`.mmb`	memory	memory	branch
`.mmi`	memory	memory	integer
`.mbb`	memory	branch	branch
`.mfb`	memory	floating point	branch
`.mmf`	memory	memory	floating point

Refer to the *Intel® Itanium(TM) Architecture Software Developer's Manual* for more information about template field encoding and instruction slot mapping.

Note:

Select the `.mlx` directive for the move long immediate instruction and for the long branch instruction. These instructions operate on 64-bit data types and are too large to fit into one of the 41-bit bundle slots. This directive selects the mlx template and inserts the instruction in slot 1 and slot 2 of the bundle.

Example below is the code that shows an explicit bundle using explicit template selection, and a stop.

Example: Bundle with Explicit Template Selection and a Stop

```
{.mmi    //use the mmi template for this
bundle
 m inst  //memory instruction
 ;;      //stop
 m inst  //memory instruction
 i inst  //integer instruction
 }
```

Instruction Groups

Itanium(TM) architecture instructions are organized in instruction groups. Each instruction group contains one or more statically contiguous instruction(s) that can execute in parallel. An instruction group must contain at least one instruction; there is no upper limit on the number of instructions in an instruction group.

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

BIAX CORPORATION	§	
	§	
V.	§	CIVIL NO. 2:05-CV-184(TJW)
	§	
INTEL CORPORATION	§	

DOCKET CONTROL ORDER

In accordance with the case scheduling conference held herein on the 20th day of December, 2005, it is hereby

ORDERED that the following schedule of deadlines is in effect until further order of this court:

May 7, 2007	Jury Selection - 9:00 a.m. in **Marshall, Texas**
April 25, 2007	Pretrial Conference - 9:00 a.m. in **Marshall, Texas**
April 10, 2007	Joint Pretrial Order, Joint Proposed Jury Instructions and Form of the Verdict.
April 20, 2007	**Motions in Limine (due three days before final Pre-Trial Conference).** Three (3) days prior to the pre-trial conference provided for herein, the parties shall furnish a copy of their respective Motions in Liming to the Court by facsimile transmission, **903/935-2295.** The parties are directed to confer and advise the Court on or before 3:00 o'clock p.m. the day before the pre-trial conference which paragraphs are agreed to and those that need to be addressed at the pre-trial conference.

April 12, 2007	Response to Dispositive Motions (including *Daubert* motions)[1] **Responses to dispositive motions filed prior to the dispositive motion deadline, including *Daubert* Motions, shall be due in accordance with Local Rule CV-7(e). Motions for Summary Judgment shall comply with Local Rule CV56.**
March 29, 2007	For Filing Dispositive Motions and any other motions that may require a hearing (including *Daubert* motions)
March 15, 2007	Mediation to be completed
February 27, 2007	Defendant to Identify Trial Witnesses
February 13, 2007	Plaintiff to Identify Trial Witnesses
February 13, 2007	Discovery Deadline
________________	30 Days after claim construction ruling Designate Rebuttal Expert Witnesses other than claims construction Expert witness report due Refer to Discovery Order for required information.
________________	15 Days after claim construction ruling Comply with P.R. 3-8.

[1] The parties are directed to Local Rule CV-7(d), which provides in part that "[i]n the event a party fails to oppose a motion in the manner prescribed herein the court will assume that the party has no opposition." Local Rule CV-7(e) provides that a party opposing a motion has **12 days, in addition to any added time permitted under Fed. R. Civ. P. 6(e),** in which to serve and file a response and any supporting documents, after which the court will consider the submitted motion for decision.

______________	15 Days after claim construction ruling Party with the burden of proof to designate Expert Witnesses other than claims construction Expert witness report due Refer to Discovery Order for required information.
November 15, 2006	Claim construction hearing 9:00 a.m., **Marshall, Texas.**
October 23, 2006	Comply with P.R. 4-5(c).
October 16, 2006	Comply with P.R. 4-5(b).
October 2, 2006	Comply with P.R. 4-5(a).
September 7, 2006	Discovery deadline–claims construction issues
August 31, 2006	Respond to Amended Pleadings
August 17, 2006	Amend Pleadings **(It is not necessary to file a Motion for Leave to Amend before the deadline to amend pleadings except to the extent the amendment seeks to add a new patent in suit. It is necessary to file a Motion for Leave to Amend after August 17, 2006).**
August 17, 2006	Comply with P.R. 4-3.
July 18, 2006	Comply with P.R. 4-2.
July 5, 2006	Privilege Logs to be exchanged by parties (or a letter to the Court stating that there are no disputes as to claims of privileged documents).
June 28, 2006	Comply with P.R. 4-1.

June 1, 2006	Deadline for damage summaries
February 17, 2006	Comply with P.R. 3-3.
January 19, 2006	Join Additional Parties
January 17, 2006	Advise Court of an agreed technical adviser
January 3, 2006	Comply with P.R. 3-1.

IT IS FURTHER ORDERED that the parties shall provide the Court the name, address, telephone number, and fax number of an agreed mediator within thirty (30) days from the date of the Scheduling Conference. If the parties are unable to agree, the Court will appoint a mediator in the above referenced case.

OTHER LIMITATIONS

1. All depositions to be read into evidence as part of the parties' case-in-chief shall be **EDITED** so as to exclude all unnecessary, repetitious, and irrelevant testimony; **ONLY** those portions which are relevant to the issues in controversy shall be read into evidence.

2. The Court will refuse to entertain any motion to compel discovery filed after the date of this Order unless the movant advises the Court within the body of the motion that counsel for the parties have first conferred in a good faith attempt to resolve the matter. See Eastern District of Texas Local Rule CV-7(h).

3. The following excuses will not warrant a continuance nor justify a failure to comply with the discovery deadline:

   (a) The fact that there are motions for summary judgment or motions to dismiss pending;

(b) The fact that one or more of the attorneys is set for trial in another court on the same day, unless the other setting was made prior to the date of this order or was made as a special provision for the parties in the other case;

(c) The failure to complete discovery prior to trial, unless the parties can demonstrate that it was impossible to complete discovery despite their good faith effort to do so.